CS 530: Theory of Computation. Based on Sipser second edition

Decidability

For convenience we use languages to represent various computational problems because we have already set up terminology for dealing with languages. For mathematical object B (be it a graph, DFA, CFG, TM, etc.) we use $\langle B \rangle$ to denote an encoding of the object as a string (so that we can give it as input to a Turing Machine). We say for $\langle B \rangle$ that B is a graph (or DFA, etc.) if $\langle B \rangle$ is a valid encoding of a graph (or DFA, etc), and that graphs is called B.

For example, the **acceptance problem** for DFAs of testing whether a particular finite automaton accepts a given string can be expressed as a language, A_{DFA}. This language contains the encodings of all DFAs together with strings that the DFAs accept.

Let $A_{DFA} = \{\langle B \rangle, w \mid B \text{ is a DFA that accepts input string } w\}$.

Theorem 1 (Theorem 4.1, page 166) *A_{DFA} is a decidable language.*

We can also prove a similar theorem for nondeterministic finite automata. Let $A_{NFA} = \{\langle B \rangle, w \mid B \text{ is a NFA that accepts input string } w\}$.

Theorem 2 (Theorem 4.2, page 167) *A_{NFA} is a decidable language.*

Similarly, we can test whether a regular expression generates a given string. Let $A_{REX} = \{\langle R \rangle, w \mid R \text{ is a regular expression that generates string } w\}$.

Theorem 3 (Theorem 4.3, page 168) *A_{REX} is a decidable language.*

In the preceding theorems we had to determine whether a finite automaton accepts a particular string. In the next theorem we determine whether a finite automaton accepts any strings at all. Let $E_{DFA} = \{\langle A \rangle \mid A$ is a DFA and $L(A) = \emptyset\}$.

Theorem 4 (Theorem 4.4, page 168) *E_{DFA} is a decidable language.*

The next theorem states that determining whether two DFAs recognize the same language is decidable. Let $EQ_{DFA} = \{\langle A \rangle, \langle B \rangle \mid A$ and B are DFAs and $L(A) = L(B)\}$.

Theorem 5 (Theorem 4.5, page 169) *EQ_{DFA} is a decidable language.*

In the proof, we construct a new DFA C from A and B, where C accepts only those strings that are accepted by either A or B but not by both. The language of C is

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B)).$$

This expression is sometimes called the **symmetric difference** of $L(A)$ and $L(B)$. Here $\overline{L(A)}$ is the complement of $L(A)$.

Here describe algorithms to determine whether a CFG generates a particular string and to determine whether the language of a CFG is empty. Let $A_{CFG} = \{\langle G \rangle, w \mid G$ is a CFG that generates string $w\}$.

Theorem 6 (Theorem 4.7, page 170) *A_{CFG} is a decidable language.*

As we did for DFAs, we can show that the problem of determining whether a CFG generates any strings at all is decidable. Let $E_{CFG} = \{\langle G \rangle \mid G$ is a CFG and $L(G) = \emptyset\}$.

Theorem 7 (Theorem 4.8, page 171) *E_{CFG} is a decidable language.*

Let $EQ_{CFG} = \{\langle G\rangle, \langle H\rangle \mid G$ and H are CFGs and $L(G) = L(H)\}$.

Now we show that every context-free language is decidable by a Turing machine.

Theorem 8 (Theorem 4.9, page 172) *Every context-free language is decidable.*

Now we turn to our first theorem that establishes the undecidability of a specific language: the problem of testing whether a Turing machine accepts a given input string. We call it A_{TM} by analogy with A_{DFA} and A_{CFG}. But, whereas A_{DFA} and A_{CFG} were decidable, A_{TM} is not. Let $A_{TM} = \{\langle M\rangle, w \mid M$ is a TM that accepts $w\}$. A_{TM} is sometimes called the **halting problem**.

Theorem 9 (Theorem 4.11, page 174) *A_{TM} is Turing recognizable, and undecidable.*

Definition 10 (Definition 4.12, page 175) *Assume that we have two sets A and B and a function f from A to B. Say that f is* ***one-to-one*** *if it never maps two different elements to the same place, that is, if $f(a) \neq f(b)$ whenever $a \neq b$. Say that f is* ***onto*** *if it hits every element of B, that is, for every $b \in B$ there is an $a \in A$ such that $f(a) = b$. Say that A and B are the* ***same size*** *if there is a one-to-one, onto function $f : A \to B$. A function that is both one-to-one and onto is called a* ***correspondence***. *In a correspondence every element of A maps to a unique element of B and each element of B has a unique element of A mapping to it. A correspondence is simply a way of pairing the elements of A with the element of B.*

Let $\mathcal{N}$ be the set of natural numbers $\{0, 1, 2, ...\}$.

Definition 11 (Definition 4.14, page 175) *A set A is **countable** if either it is finite or it has the same size as $\mathcal{N}$.*

Theorem 12 (Examples, pages 174-7) *The set of even positive integers is countable. The set of positive rational numbers $\mathcal{Q}$ is countable.*

Some infinite sets no correspondence with $\mathcal{N}$ exists. These sets are called **uncountable**. Let $\mathcal{R}$ be the set of real numbers.

Theorem 13 (Theorem 4.17, page 177) *$\mathcal{R}$ is uncountable.*

Corollary 14 (Corollary 4.18, page 178) *Some languages are not Turing-recognizable.*

In the proof, an *infinite binary sequence* is an unending sequence of 0s and 1s. Let $\mathcal{B}$ be the set of all infinite binary sequences. Let $\mathcal{L}$ be the set of all languages over alphabet Σ. Let $\Sigma^* = \{s_1, s_2, s_3, ...\}$. Each language $A \in \mathcal{L}$ has a unique sequence in $\mathcal{B}$. The ith bit of that sequence is a 1 if $s_i \in A$ and a 0 if $s_i \notin A$, which is called the **characteristic sequence** of A.

We say a language is **co-Turing-recognizable** if it is the complement of a Turing-recognizable language.

Theorem 15 (Theorem 4.22, page 181) *A language is decidable if and only if it is both Turing-recognizable and co-Turing-recognizable.*

In other words, a language is decidable if and only if both it and its complement are Turing-recognizable.

Corollary 16 (Corollary 4.23, page 182) *$\overline{A_{TM}}$ is not Turing-recognizable.*